Introduction to Evolutionary Systems

Natural language and ARtificial intelligence Group

Optimization Algorithms

- Try and find some optimal configuration of a feature vector
- Feature vectors depend on application area
- Today we'll be trying to discover the best fit curve for a mystery dataset

PSO Algorithms

- Searched by flocking particles
- Particles moved toward best ever as well as best at the current time
- Had randomness to perturb the search

Genetic Algorithms

- Based on principals of Darwinian Evolution, mainly Natural Selection and Survival of the Fittest
- Instead of flocking particles we breed them
- Only the best get to breed
- We also mutate things to make the search a bit more messy

Pseudo Code GA

```
While we haven’t found the answer
   Update the populations fitness
   Breed a new generation
End
Return the best
```

The devil is in the details (specifically the breeding)

Populations

- Populations consist of individuals
- An individual consists of two items:
  - A chromosome of numbers that correspond to a feature vector
  - A fitness value to judge how well the individual is doing

Fitness

- Fitness is a domain specific measure of how close the given individual is from the optimal solution
- Can be things like order to visit cities, weights in a neural network or coefficients in an equation

Breeding,
How does that work?

- First we need to pick some parents, and we need to pick the best ones
- Then we need to apply some genetic operator on them and put their children in the next generation

Genetic Operators

- Manipulate an individuals chromosome in some way to produce a child
- Sometimes a child is better than their parent, most of the time they're some horrible mutant
- Pick the best parents to reduce the amount of horrible mutant children

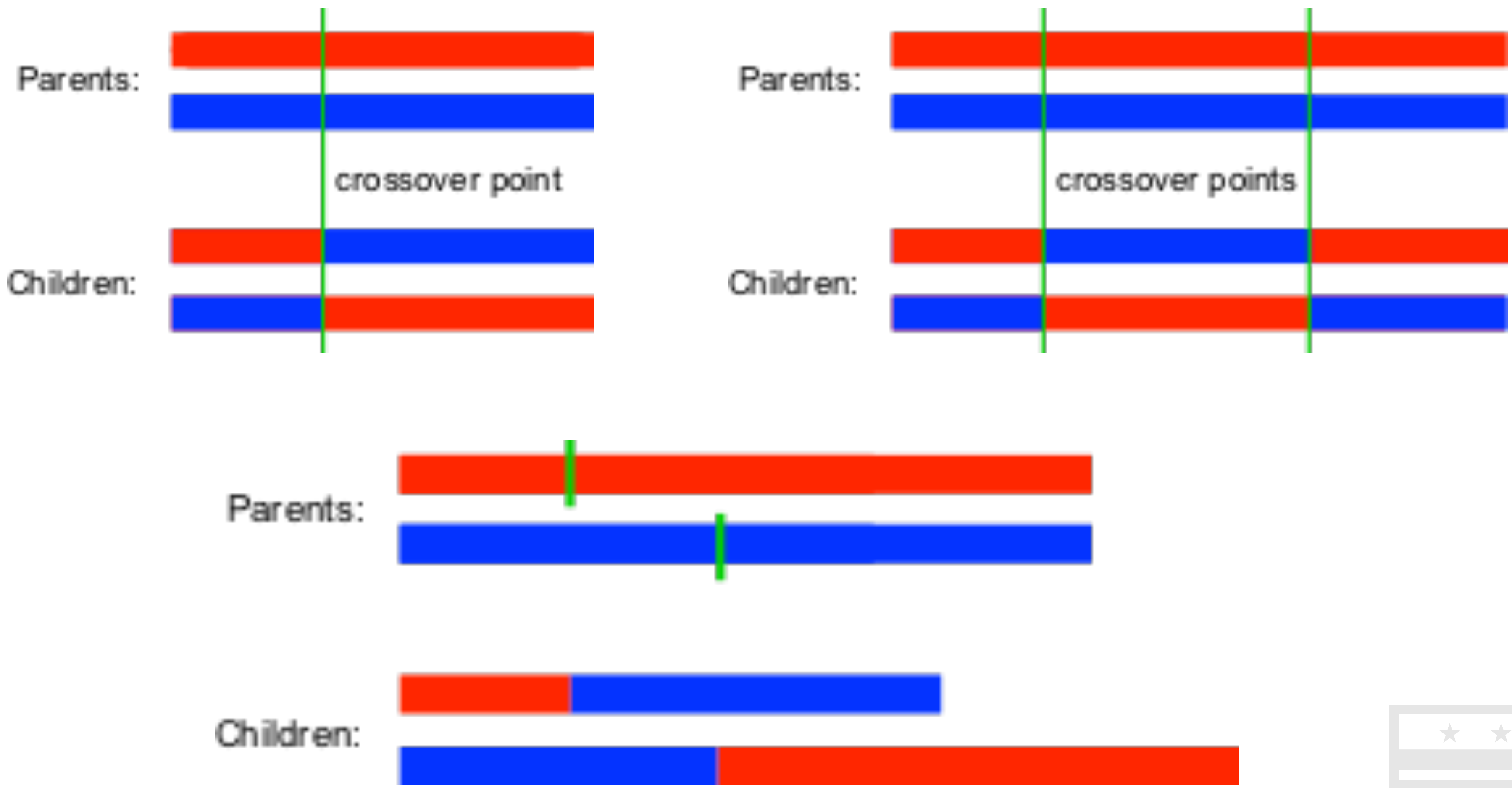
Mutation

- Point mutation is the most common genetic operator
- Picks a point on the chromosome and tweaks the value, just a little
- This tweak can make an individual better, or worse

Sexual Recombination

- Sex, it takes two parents
- Mixes parent chromosomes to create a unique child whose chromosome is a combination of both parents
- 3 major flavors: Single Point, Double Point and Cut and Splice
- We use Double Point

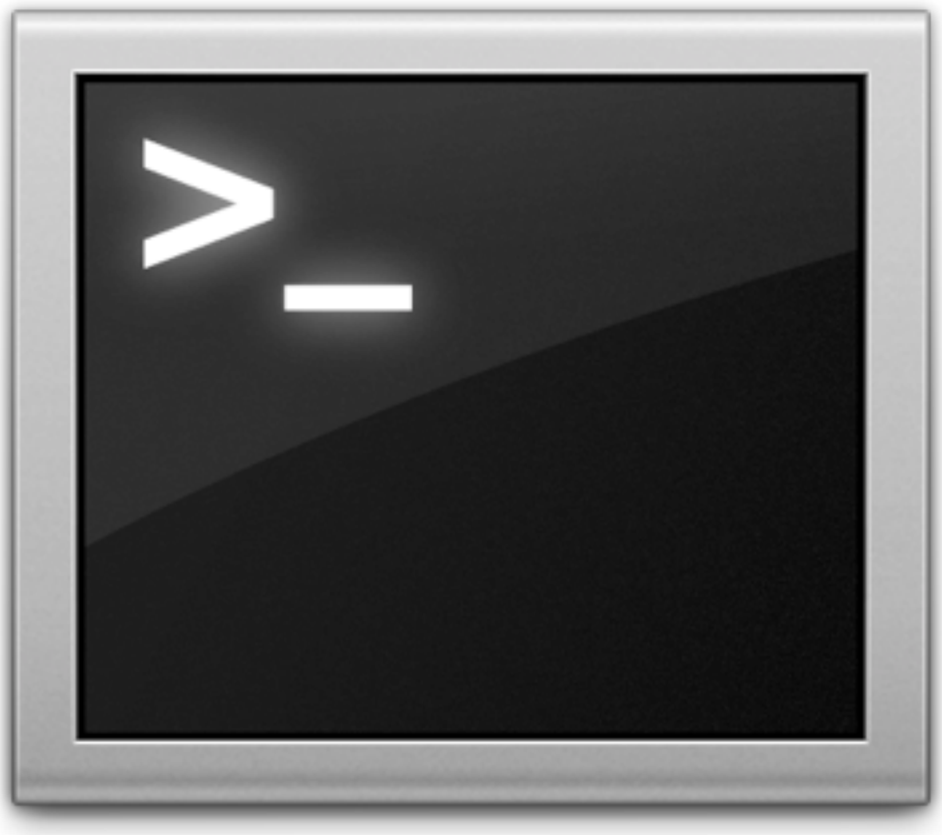
Sexual Recombination



Selection

- Many ways to do it, but you always need the best parents!
- Fitness Proportional - Individuals are picked proportional to how good they are
- Tournament - Pick N individuals at random find the best 2
- We use tournament selection

Time for some Code



Runtime Loop

```lua
local population = create_population()
local iteration = 0
local best = {}
best.chromosome = {}
best.fitness = 10000

while best.fitness > params.success
 and iteration < params.max_iterations do
    best = update_fitness(population)
    if print_stats and type(print_stats) == 'function' then
        print_stats(iteration, population, best)
    end
    population = next_generation(population)
    iteration = iteration + 1
end
return best
```

Individuals

```
individual = {
    chromosome = {1, 2, 3},
    fitness = 10000
}
```

A population is an array of these

Creating Populations

```lua
local create_population = function ()
    local dim = %params.dimension or 2
    local pop_size = %params.population_size or 100

    local population = {}
    for i=1, pop_size do
        local ind = {}
        ind.fitness = 10000
        ind.chromosome = {}
        for j=1, dim do
            tinsert(ind.chromosome, random())
        end
        tinsert(population, ind)
    end

    return population
end
```

Updating Fitness

```
local update_fitness = function (population)
    local best = population[1]
    for i=1, getn(population) do
        population[i].fitness = %fitness_func(population[i])
        if population[i].fitness < best.fitness then
            best = population[i]
        end
    end

    local new_best = {}
    new_best.fitness = best.fitness
    new_best.chromosome = {}
    for i=1, getn(best.chromosome) do
        new_best.chromosome[i] = best.chromosome[i]
    end
    return best
end
```

Next Generation

```
local next_generation = function (population)
    local next_gen = {}
    while getn(next_gen) < getn(population) do
        local parent1, parent2 = %select_parents(population)
        local x = random()
        if x > %params.mutation_rate then
            local kid = %mutation(parent1)
            tinsert(next_gen, kid)
        else
            local kid1, kid2 = %crossover(parent1, parent2)
            tinsert(next_gen, kid1)
            if getn(next_gen) < getn(population) then
                tinsert(next_gen, kid2)
            end
        end
    end
    return next_gen
end
```

Select Parents

```
local select_parents = function (population)
    local tournament = {}
    local parent1, parent2
    for i=1, %params.tournament_size or 7 do
        tinsert(tournament, population[random(getn(population))])
    end
    parent1 = tournament[1]
    parent2 = tournament[2]
    for i=1, getn(tournament) do
        if tournament[i].fitness < parent1.fitness then
            parent2 = parent1
            parent1 = tournament[i]
        end
    end

    return parent1, parent2
end
```

Mutation

```
local mutation = function (parent)
    local index = random(getn(parent.chromosome))
    local kid = {}
    kid.chromosome = {}
    kid.fitness = 10000
    for i=1, getn(parent.chromosome) do
        tinsert(kid.chromosome, parent.chromosome[i])
    end
    kid.chromosome[index] = kid.chromosome[index] + (random() - .5)
    return kid
end
```

Crossover Part 1

```
local crossover = function (parent1, parent2)
    local kid1 = {}
    kid1.chromosome = {}
    kid1.fitness = 10000

    local kid2 = {}
    kid2.chromosome = {}
    kid2.fitness = 10000

    for i=1, getn(parent1.chromosome) do
        kid1.chromosome[i] = parent1.chromosome[i]
        kid2.chromosome[i] = parent2.chromosome[i]
    end

    local index1 = random(getn(parent1.chromosome))
    local index2 = random(getn(parent2.chromosome))
    local start, stop
```

Crossover Part 2

```
    if index1 > index2 then
        start = index2
        stop = index1
    else
        start = index1
        stop = index2
    end

    for i=start, stop do
        tmp = kid1.chromosome[i]
        kid1.chromosome[i] = kid2.chromosome[i]
        kid2.chromosome[i] = tmp
    end

    return kid1, kid2
end
```

Application

- We have a mystery dataset
- We suspect it's a parabola
- Let's let the GA figure it out

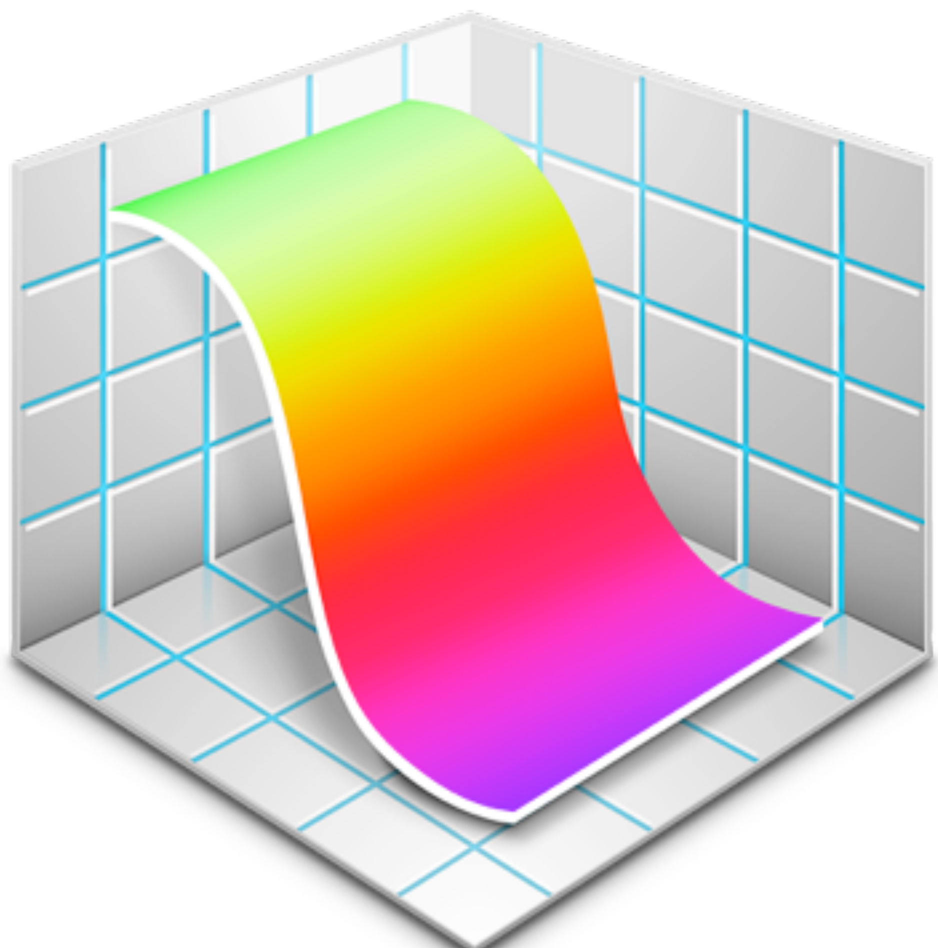
```
{5.137499, 0.18893957, -1.5235602, 0.0, 4.75962, 12.7553005, 23.98704}
```

Fitness Function

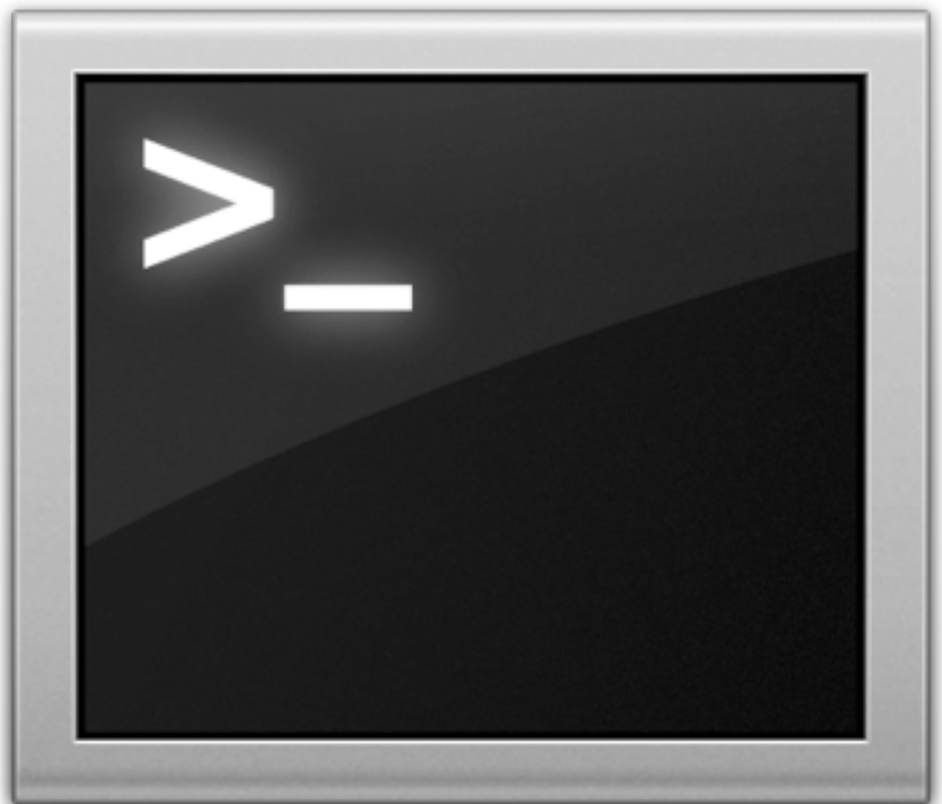
```
function fitness_func (ind)
    local test_data = {
        5.137499,
        0.18893957,
        -1.5235602,
        0.0,
        4.75962,
        12.7553005,
        23.98704
    }
    local err = 0.0
    for i=1, getn(test_data) do
        local y = (ind.chromosome[1]*i)+(ind.chromosome[2]*(i^2))
        err = err + sqrt((y - test_data[i])^2)
    end
    return err/getn(test_data)
end
```

Fitness Landscape

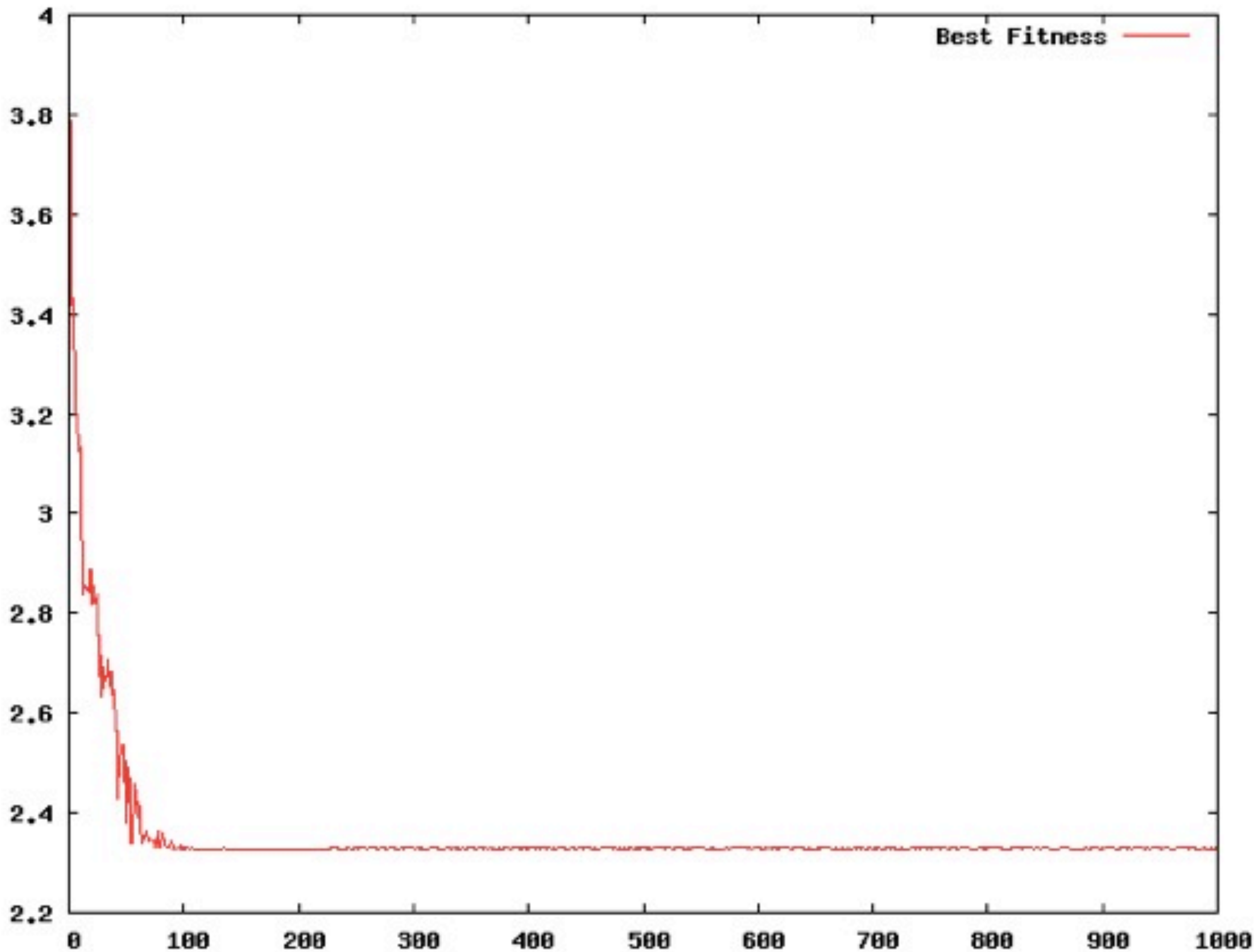
Any Ideas?



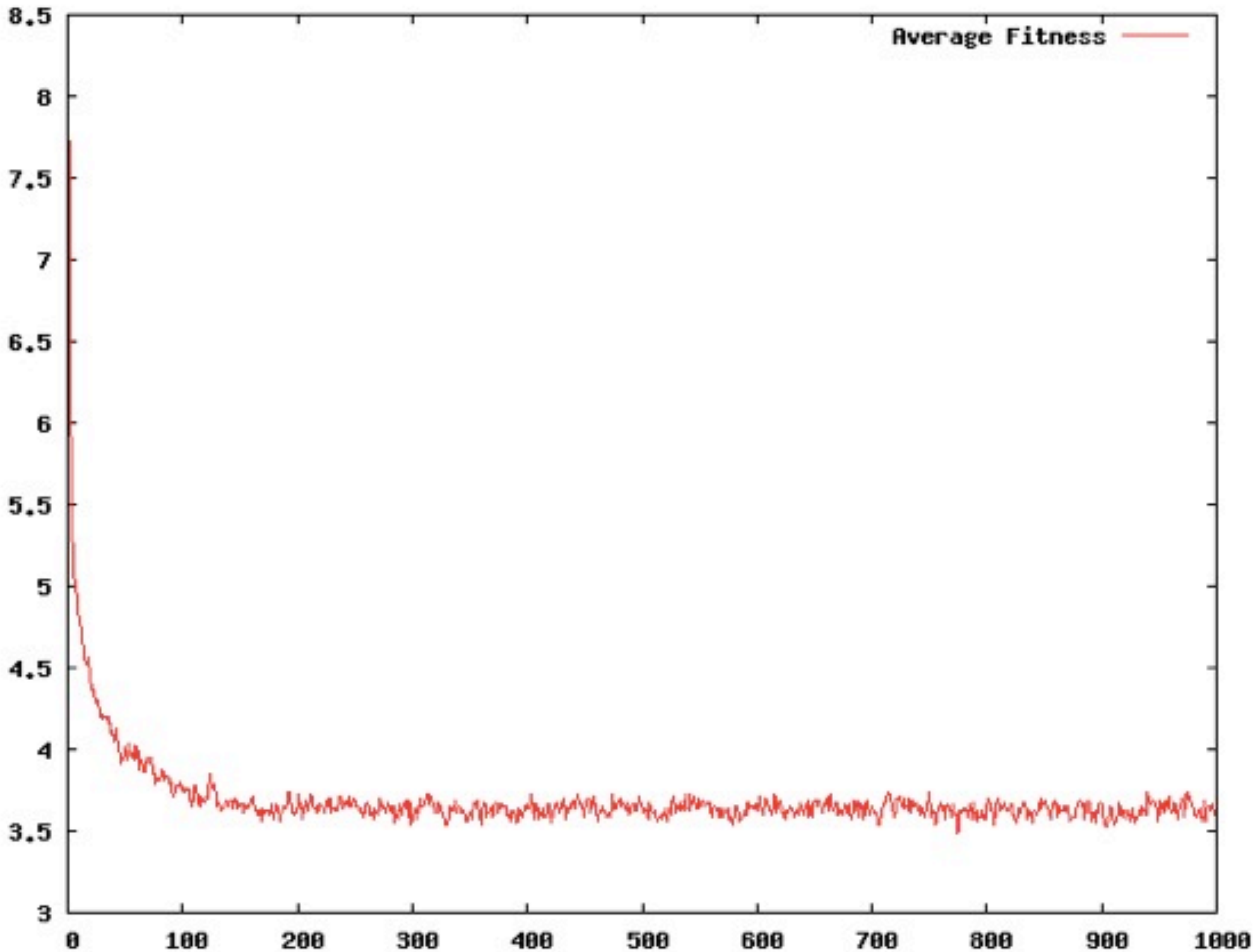
Let's Run The Code



Best Fitness Over Time



Average Fitness Over Time



Movie Time!

PSO vs GA

- The main difference between the two is the level of randomness
- PSO algorithms hone in on the answer faster than a GA
- GAs spend more time wandering due to generally high mutation rates
- Both can be tweaked to be more or less random